Linear Time Language Recognition on Cellular Automata with Restricted Communication *

Thomas Worsch
Fakultät für Informatik, Universität Karlsruhe
worsch@ira.uka.de

Abstract

It is well-known that for classical one-dimensional one-way CA (OCA) it is possible to speed up language recognition times from $(1+r)n$, $r \in \mathbb{R}_+$, to $(1+r/2)n$. In this paper we show that this no longer holds for OCA in which a cell can comminucate only one bit (or more generally a fixed amount) of information to its neighbor in each step. For arbitrary real numbers $r_2 > r_1 > 1$ in time $r_2 n$ 1-bit OCA can recognize strictly more languages than those operating in time $r_1 n$. Thus recognition times may increase by an arbitrarily large constant factor when restricting the communication to 1 bit. The results are generalized to OCA with an "average communication bandwidth" of z bits, where $z \in \mathbb{Q}_+$.

1 Introduction

The model of 1-bit CA results from the standard definition by restricting the amount of information which can be transmitted by a cell to its neighbors in one step to be only 1 bit. We call this the communication bandwidth.

Probably the first paper investigating 1-bit CA is the technical report by Mazoyer (1989) where it is shown that even with this model solutions of the FSSP in optimal time are possible. More recently Umeo (1998) has described 1-bit CA for several one- and two-dimensional problems (e.g. generation of Fibonacci sequences and determining whether two-dimensional patterns are connected) which again are running in the minimum time possible.

Therefore immediately the questions arises about the consequences of the restriction to 1-bit information flow in the general case.

In Section 2 basic definitions are given and it is proved that each CA with s states can be simulated by a 1-bit CA with a slowdown by a factor of at most $\lceil \log s \rceil$. This seems to be some kind of folklore, but we include the proof for the sake of completeness and reference in later sections.

In Section 3 it is shown that for one-way CA (OCA) in general there *must* be a slowdown. More specifically there is a very fine hierarchy with an uncountable number of distinct levels (order isomorphic to the real numbers greater than 1) within the class of languages which can be recognized by 1-bit OCA in linear time.

Generalizations are possible in different directions. In Section 4 we mention preliminary results extending the ones presented here in different directions.

*This is technical report 25/98 of the Department of Informatics, University of Karlsruhe. It is also available at `http://liinwww.ira.uka.de/~worsch/research/papers/` on the WWW.

2 Simulation of k-bit CA by 1-bit CA

A deterministic CA is determined by a finite set of states Q, a neighborhood $N' = N \cup \{0\}$ and a local rule. (For a simpler notation below, we assume that $N = \{n_1, \ldots, n_{|N|}\}$ does not contain $\{0\}$). The local rule of C is of the form $\tau : Q \times Q^N \to Q$, i.e. each cell can has the *full* information on the states of its neighbors.

In a k-bit CA B, each cell only gets k bits of information about the state of each neighbor. To this end there are functions $b_i : Q \to \mathbb{B}^k$ specified, where $\mathbb{B} = \{0, 1\}$. If a cell is in state q then $b_i(q)$ are the bits observed by neighbor n_i. We allow different bits to be seen by different neighbors. The local transformation of B is of the form $\tau : Q \times (\mathbb{B}^k)^N \to Q$.

Given a configuration $c : \mathbb{Z} \to Q$ and its successor configuration c' the new state of a cell i is $c'_i = \tau(c_i, b_1(c_{i+n_1}), \ldots, b_{|N|}(c_{i+n_{|N|}})$.

As usual, for the recognition of formal languages over an input alphabet A one chooses $Q \supset A$ and a set of accepting final states $F \subseteq Q \setminus Q$. In the initial configuration for an input $x_1 \cdots x_n \in A^n$ cell i is in state x_i for $1 \leq i \leq n$ and all other cells are in a quiescent state q (satisfying $\tau(q, q^N) = q$). A configuration c is accepting iff $c_1 \in F$.

Given a k-bit CA C one can construct a 1-bit CA C' with the same neighborhood simulating C in the following sense: Each configuration c of C is also a legal configuration of C', and there is a constant l (independent of c) such that if c' is C's successor configuration of c then C' when starting in c reaches c' after l steps. The basic idea is to choose representations of states by binary words and to transmit them bit by bit to the neighbors before doing a "real" state transition.

Let $\mathbb{B}^{\leq i}$ denote $\mathbb{B}^0 \cup \cdots \cup \mathbb{B}^i$. Denote by $b_{i,j}(q)$ the j-th bit of $b_i(q)$, i.e. $b_i(q) = b_{i,k}(q) \cdots b_{i,1}(q)$.

2.1 Algorithm. As the set of states of C' choose $Q' = Q \times (\mathbb{B}^N)^{\leq k-1}$; i.e. each state q' consists of a $q \in Q$ and binary words $v_1, \ldots, v_{|N|}$ of identical length j for some $0 \leq j \leq k-1$. For each $q \in Q$ identify $(q, \varepsilon, \ldots, \varepsilon)$ with q so that Q can be considered a subset of Q'. (Here, ε is the empty word.) For $j \leq k-1$ and a $q' = (q, v_1, \ldots, v_{|N|}) \in Q \times (\mathbb{B}^N)^j$ define $b'_i(q') = b_{i,j+1}(q)$, where the b'_i are the functions describing the bit seen by neighbor n_i in C'.

The local transformation τ' of C' is defined as follows:

- If the length j if all v_e is $< k-1$ then $\tau'((q, v_1, \ldots, v_{|N|}), x_1, \ldots, x_{|N|}) = (q, x_1 v_1, \ldots, x_{|N|} v_{|N|})$.
- If the length j if all v_e is $= k-1$ then $\tau'((q, v_1, \ldots, v_{|N|}), x_1, \ldots, x_{|N|}) = (\tau(q, x_1 v_1, \ldots, x_{|N|} v_{|N|}), \varepsilon, \ldots, \varepsilon)$.

□

The above construction shows that the following lemma holds:

2.2 Lemma. *A k-bit* CA *can be simulated by a* 1*-bit* CA *with the same neighborhood and slowdown k.*

Since the states of a set Q can be unambiguously represented as binary words of length $\lceil \log_2 |Q| \rceil$, it is straightforward to see:

2.3 Corollary. *Each* CA *with s states can be simulated by a* 1*-bit* CA *with slowdown $\lceil \log_2 s \rceil$ having the same neighborhood and identical functions b_i for all neighbors.*

It should be observed that the above slowdown happens if the bit visible to other cells is the *same* for all neighbors. One could wonder whether the slowdown is always less if different bits are sent to different neighbors. However this is not the case. The proofs below for the lower bounds do not specifically make any use of the fact that all neighbors are observing the same bit; they work even if there were $|N|$ (possibly different) functions b_i for the neighboring cells.

On the other hand one should note that for certain CA there is a possibility for improvement, i.e. conversion to 1-bit CA with a smaller slowdown: Sometimes it is already known that neighbors do not need the full information about each state. In a typical case the set of states might be the Cartesian product of some sets and a neighbor only needs to know one component, as it is by definition the case in so-called partitioned CA. It is then possible to apply a similar construction as above, but only to that component. Since the latter can be described with less bits than the whole state, the construction results in a smaller slowdown.

We will make use of this and a related trick in Section 3.3.

3 A linear-time hierarchy for 1-bit OCA

For a function $f : \mathbb{N}_+ \to \mathbb{N}_+$ denote by $\mathrm{OCA}_k(f(n))$ the family of languages which can be recognized by k-bit OCA in time $f(n)$. In this section we want to prove:

3.1 Theorem. *For all real numbers $1 < r_1 < r_2$ holds:*

$$\mathrm{OCA}_1(r_1 n) \subsetneqq \mathrm{OCA}_1(r_2 n)$$

We will proceed in 3 major steps.

3.1 An infinite hierarchy

Let A_m be an input alphabet with exactly $m = 2^l - 1$ symbols. Hence l bits are needed to describe one symbol of $A_m \cup \{\Box\}$, where $\Box$ is the quiescent state. The case of alphabets with an arbitrary number of symbols will be considered later.

Denote by L_m the set $\{vv^R \mid v \in A_m^+\}$ of all palindromes of even length over A_m.

3.2 Lemma. *Each* 1*-bit OCA recognizing L_m needs at least time $(l-\varepsilon)n$ for every $\varepsilon > 0$.*

3.3 Proof. Consider a 1-bit OCA C recognizing L_m and an input length n. Denote by t the worst case computation time needed by C for inputs of length n.

Consider the boundary between cells k and $k+1$, $1 \le k < n$, which separates a left and a right part of an input. The computations in the left part are completely determined by the corresponding part of the input and the sequence B_r of bits received by cell k from the right during time steps $1, \ldots, t-k$. There are exactly 2^{t-k} such bit sequences. On the other hand there are $m^{(n-k)} = (2^l - 1)^{(n-k)}$ right parts of inputs of length n.

Assume that $2^{t-k} < (2^l - 1)^{(n-k)}$. Then there would exist two different words v_1 and v_2 of length $n-k$ resulting in the same bit string received by cell k during *any* computation for an input of one of the forms vv_1 or vv_2. Since we are considering

OCA, the bit string is independent of any symbols to the left of cell $k+1$. Therefore C would either accept or reject *both* inputs v_1v_1 or v_1v_2, although exactly one of them is in L_m. Contradiction.

Therefore $2^{t-k} \geq (2^l-1)^{(n-k)}$. For sufficiently large n there is an arbitrarily small ε'' such that this implies $2^{t-k} \geq 2^{(l-\varepsilon'')(n-k)}$, i.e. $t-k \geq (l-\varepsilon'')(n-k)$, i.e. $t \geq ln+k-lk-\varepsilon''n+\varepsilon''k$. For an arbitrarily chosen $\varepsilon' > 0$ consider the case $k = \varepsilon' n$ (for sufficiently large n). One then gets $t \geq ln+\varepsilon' n - l\varepsilon' n - \varepsilon'' n + \varepsilon''\varepsilon' n = ln - (\varepsilon'(l-1-\varepsilon'')+\varepsilon'')n$. If ε' and ε'' are chosen sufficiently small this is larger than $ln - \varepsilon n$ for a given ε. ■

3.4 Lemma. *L_m can be recognized by 1-bit OCA in time $(l+1)n + \mathrm{O}(1)$.*

3.5 Proof. Algorithm 3.6 below describes a $(l+1)$-bit OCA recognizing the language in time $n+\mathrm{O}(1)$. Hence the claim follows from Lemma 2.2. ■

3.6 Algorithm. We describe a $(l+1)$-bit OCA recognizing L_m. The set of states can be chosen to be of the form $Q_m = A_m \cup A_m \times (A_m \cup \{\Box\}) \times \mathbb{B}^2$. The local rule mapping the state q_c of a cell and the state q_r of its neighbor to $\tau(q_c, q_r)$ is chosen as follows. For the first step, with $a, b' \in A_m$:

$$\tau(a, b') = (a, b', 1, 1)$$

For later steps:

$$\tau((a, a', x, x'), (b, b', y, y')) = (a, b', x' \wedge [a = a'], y)$$

where $[a = a']$ is 1 if the symbols are equal and 0 otherwise. As can be seen immediately, the only information needed from the right neighbor is one symbol b' and one bit y. Hence an $(l+1)$-bit OCA can do the job.

A closer look at the local rule reveals that the OCA above indeed recognizes palindromes in time $n+\mathrm{O}(1)$ if one chooses as the set of final states $A_m \times \{\Box\} \times \{1\} \times \mathbb{B}$ (see Vollmar and Worsch (1995) for details). Hence L_m can also be recognized by 1-bit OCA in time $(l+1)n+\mathrm{O}(1)$. □

The upper bound of the previous lemma is not very close to the lower bound of Lemma 3.2, and it is not obvious how to improve at least one of them.

3.2 Reducing the gap to $(l \pm \varepsilon)n$

We will now define variants of the palindrome language for which gaps between upper and lower bound can be proved to be very small.

We will use vectors of length r of symbols from an alphabet A as symbols of a new alphabet A'. Although a vector of symbols is more or less the same as a word of symbols, we will use different notations for both concepts in order to make the construction a little bit clearer. Denote by $M^{\langle r \rangle}$ the set of vectors of length r of elements from a set M and by A^r the set of words of length r consisting of symbols from A. The obvious mapping $\langle x_1, \ldots, x_r \rangle \mapsto x_1 \cdots x_r$ induces a monoid homomorphism $h : (A^{\langle r \rangle})^* \to (A^r)^* \subseteq A^*$.

3.7 Definition. For integers $m \geq 1$ and $r \geq 1$ let

$$L_{m,r} = \{vh(v)^R \mid v \in (A_m^{\langle r \rangle})^+\}$$

$L_{m,r}$ is a language over the alphabet $A_m^{\langle r\rangle} \cup A_m$. The words in $L_{m,r}$ are still more or less palindromes where in the left part of a word groups of r elements from A_m are considered as *one* symbol. As a special case one has $L_{m,1} = L_m$ as defined earlier.

3.8 Lemma. *For each $\varepsilon > 0$ there is an $r \geq 1$ such that each 1-bit OCA recognizing $L_{m,r}$ needs at least time $(l - \varepsilon)n$.*

A proof can be given analogously to 3.3 above. One only has to observe that the border between cells k and $k+1$ must not lie within "the left part v" of an input. Therefore for small ε one must choose a sufficiently large r, e.g. $r > 1/\varepsilon$, to make sure that $|v| < \varepsilon |vh(v)^R|$.

Thus for sufficiently large r although $\lceil \log_2 |A_m| \rceil \cdot n$ is not a lower bound on the recognition time of $L_{m,r}$ by 1-bit OCA, it is "almost".

3.9 Lemma. *For each $\varepsilon > 0$ and $r = 1/\varepsilon$ the language $L_{m,r}$ can be recognized by a 1-bit OCA in time $(l + \varepsilon)n + \mathrm{O}(1)$.*

Thus for sufficiently large r although $\lceil \log_2 |A_m| \rceil \cdot n$ is not an upper bound on the on the achievable recognition time on 1-bit OCA, it is "almost".

For the proof we use a construction similar to Algorithm 3.6.

3.10 Algorithm. The CA uses a few additional steps before and after the check for palindromes, where the check itself also has to be adapted to the different form of inputs.

- In the first step each cell sends one bit to its left neighbor indicating whether its input symbol is from A_m or $A_m^{\langle r\rangle}$. Thus, if the input is *not* in $(A_m^{\langle r\rangle})^* A_m^*$ this is detected by at least one cell and an error indicator is stored locally. It will be used later.
- One may therefore assume now that the input is of the indicated form, and we will call cells with an input symbol from A_m the "right" cells and those with a symbol from $A_m^{\langle r\rangle}$ the "left" cells.

  After the first step the rightmost of the left cells has indentified itself.
- With the second step an algorithm for palindrome checking is started. The modifications with respect to Algorithm 3.6 are as follows:

  - Each cell is counting modulo $lr + 1$ in each step. (This doesn't require any communication.)
  - During the first lr steps of a cycle the right cells are shifting r symbols to the left. In the $(lr+1)$-st step they do not do anything.
  - During the first lr steps of a cycle the left cells are also shifting r symbols to the left. In addition they are accumulating what they receive in registers. In step lr step they are comparing whether the register contents "match" their own input symbol, and in step $lr + 1$ they are sending the result of the comparison, combined with the previously received comparison bit to their left neighbor.

  One should observe that the last point is the basic trick: the comparison bit has not to be transported one cell to the left each time a symbol has been received, but only every r symbols. Thus by increasing r the fraction of time needed for transmitting these bits can be made arbitrarily small.

- All the algorithms previously described have the following property: The part of the time space diagram containing all informations which are needed for the decision whether to accept or reject an input has the form of a triangle. Its longest line is a diagonal with some slope $n/t(n)$ (or $t(n)/n$ depending on how you look at it) leading from the rightmost input cell the leftmost one. Furthermore every cell can know when it has done its job because afterwards it only receives the encodings of the quiescent state.
- Therefore the following signal can be implemented easily: It starts at the rightmost input cell and collects the results of the checks done in the very first step. It is moved to the left immediately after a cell has transmitted at least one (encoding of the) quiescent state in a $(lr+1)$-cycle. Thus this signal causes only one additional step to the overall recognition time.

Since the above algorithm needs $lr+1$ steps per r input symbols from A_m and since the rightmost r symbols have to travel approximately n cells far, the total running time is $n\cdot(lr+1)/r+\mathrm{O}(1)$, i.e. $(l+1/r)n+\mathrm{O}(1)$ as required. □

From the Lemmata 3.8 and 3.9 one can immediately deduce the following:

3.11 Corollary. *For each integer constant c the set of languages which can be recognized by 1-bit OCA in time cn is strictly included in the the set of languages which can be recognized by 1-bit OCA in time $(c+2)n$.*

This has to be contrasted with unlimited OCA where there is no such infinite hierarchy within the family of languages which can recognized in linear time. One therefore gets the following picture:

$$\begin{array}{ccccccccc}
\mathrm{OCA}_1(2n) & \subsetneq & \mathrm{OCA}_1(4n) & \subsetneq & \mathrm{OCA}_1(6n) & \subsetneq & \mathrm{OCA}_1(8n) & \subsetneq & \cdots \\
\cap\!\!\!\!\neq & & \cap\!\!\!\!\neq & & \cap\!\!\!\!\neq & & \cap\!\!\!\!\neq & & \\
\mathrm{OCA}(2n) & = & \mathrm{OCA}(4n) & = & \mathrm{OCA}(6n) & = & \mathrm{OCA}(8n) & = & \cdots
\end{array}$$

Figure 1: A hierarchy for 1-bit OCA.

In the top row one uses the fact that for each $i \geq 1$

$$\mathrm{OCA}_1(2in) \subseteq \mathrm{OCA}_1((2i+1-\varepsilon)n) \subsetneq \mathrm{OCA}_1((2i+1+\varepsilon)n) \subseteq \mathrm{OCA}_1((2i+2)n)$$

and for each column one has to observe

$$\mathrm{OCA}_1(2in) \subsetneq \mathrm{OCA}_1((2i+2)n) \subseteq \mathrm{OCA}((2i+2)n) = \mathrm{OCA}(2in)\ .$$

3.3 There are small gaps everywhere

Finally we will prove now that a small increase of the linear-time complexity already leads to an increased recognition power not only around $(r\pm\varepsilon)n$ for natural numbers r, but for all real numbers $r>1$. Since the rational numbers are dense in $\mathbb{R}$ it suffices to prove the result for $r\in\mathbb{Q}$.

The basic idea is the following: The number l playing an important role in the previous sections is something like an average number of bits needed per symbol. What we want to achieve below is an average number r of bits needed per symbol.

Assume that an arbirtrary rational number $r > 1$ has been fixed as well as the relatively prime natural numbers x and $y < x$ such that $r = x/y$. Then the above is more or less equivalent to saying that one needs x bits for every y symbols.

Therefore choose the smallest m such that $2^x < m^y$ and a set M of $2^x - 1$ different words from A_m^y. In order to define the languages $L'_{m,r}$ to be used later we start with the languages $L_{m',r}$ considered in the previous section, where $m' = 2^x - 1$. Denote by $g_{x,y}$ a one-to-one mapping $g_{x,y} : A_{m'} \to M$ which is extended to words of r-tuples by considering it as a function $g : (A_{m'}^{\langle r \rangle})^* \to ((A_m^{\langle r \rangle})^y)^*$ mapping each r-tuple of symbols is a word of length y of r-tuples of symbols. Now choose

$$L_{x,y,m,r} = g_{x,y}(L_{m',r})$$

For an example assume that we are interested in the case $r = 1/3$. Then $m = 2$ is the smallest integer such that $2^1 < m^3$.

Analogously to the previous sections we will proceed now by showing that $(x/y - \varepsilon)n$ is a lower bound on the recognition time of $L'_{m,1/\varepsilon}$ and that time $(x/y + \varepsilon)n$ suffices.

3.12 Lemma. *For each $\varepsilon > 0$ there is an $r \geq 1$ such that each 1-bit OCA recognizing $L_{x,y,m,r}$ needs at least time $(x/y - \varepsilon)n$.*

3.13 Proof. Straightforward. ∎

3.14 Lemma. *For each $\varepsilon > 0$ and $r = 1/\varepsilon$ the language $L_{x,y,m,r}$ can be recognized by a 1-bit OCA in time $(x/y + \varepsilon)n + \mathrm{O}(1)$.*

3.15 Proof. Straightforward. ∎

As a consequence one gets that there is an uncountable set of families of languages ordered by proper inclusion which is order isomorphic to the real numbers greater than 1 as already claimed at the beginning of this section:

3.16 Proof. (of Theorem 3.1) Choose a rational number x/y and an $\varepsilon > 0$ such that $r_1 < x/y - \varepsilon < x/y + \varepsilon < r_2$. From Lemmata 3.12 and 3.14 follows that there is a language in $\mathrm{OCA}_1(x/y + \varepsilon) \setminus \mathrm{OCA}_1(x/y - \varepsilon)$ which is then also a witness for the properness of the above inclusion. ∎

Therefore the hierarchy depicted in Figure 1 can be generalized to the following, where r_1 and r_2 are arbitrary real numbers satisfying $1 < r_1 < r_2$:

$$\begin{array}{ccccccc}
\cdots & \subsetneq & \mathrm{OCA}_1(r_1 n) & \subsetneq \cdots \subsetneq & \mathrm{OCA}_1(r_2 n) & \subsetneq & \cdots \\
 & & \subsetneq & & \subsetneq & & \\
\cdots & = & \mathrm{OCA}(r_1 n) & = \cdots = & \mathrm{OCA}(r_2 n) & = & \cdots
\end{array}$$

Figure 2: The very fine hierarchy for 1-bit OCA.

4 Outlook

The above results can be generalized in several ways. On is to consider z-bit OCA where $z \in \mathbb{Q}$ is a *rational* number. For a suitable definition of this model one can then show that for real numbers $1 < r_1 < r_2$ and $0 < z_1 < z_2$ the following inclusions are all proper:

$$\begin{array}{ccccccc}
 & & \vdots & & \vdots & & \\
 & & \subsetneq & & \subsetneq & & \\
\cdots \subsetneq & \mathrm{OCA}_{z_1}(r_1 n) & \subsetneq \cdots \subsetneq & \mathrm{OCA}_{z_1}(r_2 n) & \subsetneq \cdots \\
 & & \subsetneq & & \subsetneq & & \\
 & & \vdots & & \vdots & & \\
 & & \subsetneq & & \subsetneq & & \\
\cdots \subsetneq & \mathrm{OCA}_{z_2}(r_1 n) & \subsetneq \cdots \subsetneq & \mathrm{OCA}_{z_2}(r_2 n) & \subsetneq \cdots \\
 & & \subsetneq & & \subsetneq & & \\
 & & \vdots & & \vdots & & \\
 & & \subsetneq & & \subsetneq & & \\
\cdots = & \mathrm{OCA}(r_1 n) & = \cdots = & \mathrm{OCA}(r_2 n) & = \cdots
\end{array}$$

Figure 3: A very fine hierarchy for OCA with restricted communication bandwidth.

This will be shown in a forthcoming paper.

The other obvious extension is to two-way CA (and later higher dimensional CA).

It si not too difficult to see, that L_m can be recognized by 1-bit CA in time $(l+2)n/2$, but it cannot be recognized by 1-bit OCA in time $(l-\varepsilon)n$. This is a gap of $(l-\varepsilon)n - (l+2)n/2 = (l-2-2\varepsilon)n/2$ which can be made *arbitrarily large*! In other words:

4.1 Lemma. *For each constant $c > 1$ there are languages for which 1-bit two-way CA can be faster than any OCA recognizing it by a factor of c.*

4.2 Corollary. *For no constants $r > 1$ and $c > 1$ is $\mathrm{CA}_1(rn) \subseteq \mathrm{OCA}_1(crn)$.*

We also have preliminary results analogously e.g. to Lemma 3.2 for two-way CA, but still with a large gap between classes which can be shown to be related by proper inclusion. We hope to reduce the gap in a further work.

5 Conclusion

It has been shown that for all real numbers $r > 1$ and $\varepsilon > 0$ there are problems which can be solved on 1-bit OCA in time $(r+\varepsilon)n$, but not in time rn. As a consequence there are problems the solution of which on 1-bit OCA *must* be slower than on unlimited OCA by a factor of at least r.

It is therefore interesting, and in some way surprising, that certain problems which are considered to be nontrivial, e.g. the FSSP, can solved on 1-bit CA without any loss of time.

Two-way CA with the ability to communicate 1 bit of information in each direction are more powerful than one-way CA with the ability to communicate k bit in one direction. For certain formal languages the latter have to be slower by a constant factor which cannot be bounded.

Acknowledgements

The author would like to thank Hiroshi Umeo for valuable hints concerning preliminary versions of this paper and for asking the right questions. Interesting discussions with Thomas Buchholz and Martin Kutrib during IFIPCA 98 were also helpful.

References

Mazoyer, J. (1989). A minimal time solution to the firing squad synchronization problem with only one bit of information exchanged. Technical Report TR 89-03, Ecole Normale Supérieure de Lyon, Lyon.

Umeo, H. (1998). Cellular algorithms with 1-bit inter-cell communications. In Thomas Worsch and R. Vollmar, editors, *MFCS'98 Satellite Workshop on Cellular Automata*, pages 93–104.

Vollmar, R. and Worsch, Th. (1995). *Modelle der Parallelverarbeitung – eine Einführung*. Teubner, Stuttgart.